




UNIVERSITY OF COPENHAGEN

Learning on Graphs with GNNs

NORA Summer School on Geometric Deep Learning

Raghavendra Selvan

Assistant Professor (TT)
Dept. of Computer Science (ML Section)
University of Copenhagen
Pioneer Centre for Artificial Intelligence, Denmark

raghav@di.ku.dk
✉  /raghavian
<https://raghavian.github.io>

June 12, 2024



Lecture-2: Overview

I Overview of GNN Landscape

- Learning on Graphs
- GraphSAGE
- Jumping Knowledge Networks
- Graph Isomorphism Network
- Graph Attention Network
- Relational GNNs



Lecture-2: Overview

I Overview of GNN Landscape

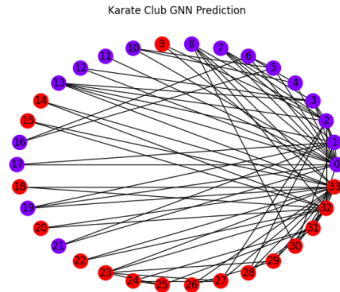
- Learning on Graphs

- GraphSAGE
- Jumping Knowledge Networks
- Graph Isomorphism Network
- Graph Attention Network
- Relational GNNs



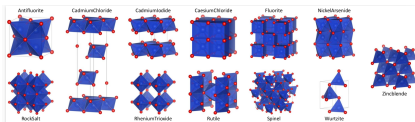
Transductive Learning on Graphs

- Transductive learning involves predicting labels for existing nodes in a graph.
- **Example:** Semi-supervised node classification on a citation network.
- Usually, $|G| = 1, N \gg F$



Inductive Learning on Graphs

- Inductive learning involves making predictions for new, unseen nodes in a graph.
- **Example:** Predicting properties of new molecules in a molecular graph.
- Usually, $|G|, N \gg F$



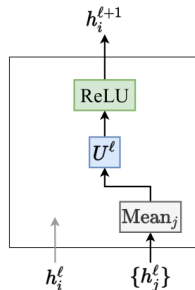
Revisit Deep GCN

Recall, that one layer of GCN is

$$\mathbf{h}_i^{(\ell+1)} = \sigma \left(\frac{\Theta^{(\ell)}}{|\mathcal{N}_i|} \sum_{j \in \mathcal{N}_i} \mathbf{h}_j^{(\ell)} \right) \text{ Or,}$$

$$\mathbf{h}_i^{(\ell+1)} = \text{UPDATE}^{(\ell)} \left(\mathbf{h}_i^{(\ell)}, \text{AGGREGATE} \left(\{ \mathbf{h}_j^{(\ell)}, \forall j \in \mathcal{N}_i \} \right) \right)$$

As a generalized message passing algorithm (we will use this interchangeably)



How deep can GCNs go?



Over-Smoothing in GNNs

- After several iterations of message passing, representations for all nodes become similar
- Common in basic GNN models
- Problematic, as it makes building deeper GNN models impossible
- Deep GNNs are important to leverage longer-range dependencies
- Formalized by defining influence of each node's input feature on final layer of all other nodes
- Examine the magnitude of corresponding Jacobian matrix:

$$l_K(i, j) = \mathbf{1}^T \left(\frac{\partial \mathbf{h}_j^{(K)}}{\partial \mathbf{h}_i^{(0)}} \right)$$

- Xu et al. 2018 prove the following: $l_K(i, j) \propto p_K(i|j)$, where, $p_K(i|j)$ is prob. of visiting node j on a length- K random walk starting from node i .
- Note that it only takes $k = \mathcal{O}(\log(|\mathcal{V}|))$ steps for random walk starting from any node to converge to an almost uniform distribution

How to overcome this? Any inspirations from CNNs?



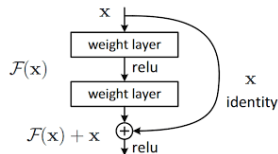
Why Skip Connections Help in Feed-Forward CNNs

- Consider a deep neural network with L layers where each layer ℓ has an input \mathbf{x}_ℓ and produces an output \mathbf{y}_ℓ .
- Without skip connections:

$$\mathbf{y}_\ell = \mathcal{F}_\ell(\mathbf{x}_\ell)$$

- With skip connections:

$$\mathbf{y}_\ell = \mathcal{F}_\ell(\mathbf{x}_\ell) + \mathbf{x}_\ell$$



Gradient During Backpropagation

- Without skip connections:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{x}_\ell} = \frac{\partial \mathcal{L}}{\partial \mathbf{y}_\ell} \cdot \frac{\partial \mathbf{y}_\ell}{\partial \mathbf{x}_\ell} = \frac{\partial \mathcal{L}}{\partial \mathbf{y}_\ell} \cdot \frac{\partial \mathcal{F}_\ell(\mathbf{x}_\ell)}{\partial \mathbf{x}_\ell}$$

- With skip connections:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{x}_\ell} = \frac{\partial \mathcal{L}}{\partial \mathbf{y}_\ell} \cdot \left(\frac{\partial \mathcal{F}_\ell(\mathbf{x}_\ell)}{\partial \mathbf{x}_\ell} + \mathbf{I} \right)$$

- The identity matrix \mathbf{I} helps maintain the gradient signal, addressing the vanishing gradient problem.



What about skip connections inspired from CNNs?

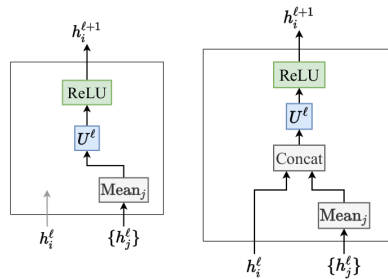
- GCN aggregates a node and its neighbours simultaneously:

$$\mathbf{h}_i^{(\ell+1)} = \sigma \left(\frac{\Theta^{(\ell)}}{|\mathcal{N}_i|} \sum_{j \in \mathcal{N}_i} \mathbf{h}_j^{(\ell)} \right)$$

- Skip the node's representation to after neighbour aggregation:

$$\mathbf{h}_i^{(\ell+1)} = \sigma \left(\Theta^{(\ell)} \left(\mathbf{h}_i^{(\ell)} + \frac{1}{|\mathcal{N}_i|} \sum_{j \in \mathcal{N}_i} \mathbf{h}_j^{(\ell)} \right) \right)$$

- Similar to skip-/residual- connections in CNNs.



Skip connections help only to some extent.

Issue of smoothing into subsequent layers persists. We will revisit this later.

Lecture-2: Overview

I Overview of GNN Landscape

- Learning on Graphs
 - **GraphSAGE**
 - Jumping Knowledge Networks
 - Graph Isomorphism Network
 - Graph Attention Network
 - Relational GNNs



GCNs and Long-range Neighbourhood

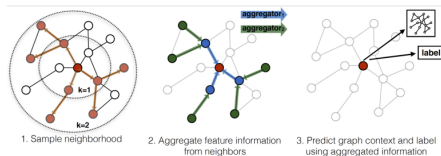
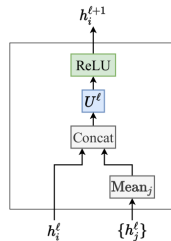
- K -layered GCN aggregates information from K hops away
- Increasing K results in over-smoothing
- GCNs are useful in transductive settings; difficult to generalize across graph structures

Can we use node features to learn the topology of neighbourhood and distribution of node features?



GraphSAGE: Sample and Aggregate

- Instead of training distinct embedding vector for each node, learn set of aggregator functions
- Each aggregator function aggregates information from different hops away
- Fixed size neighbourhood is uniformly sampled to reduce computational complexity
- $\mathbf{h}_i^{(\ell)} = \sigma \left(\Theta^{(\ell)} \text{CONCAT} \left(\mathbf{h}_i^{(\ell-1)}, \frac{1}{|\mathcal{N}_i|} \sum_{j \in \mathcal{N}_i} \mathbf{h}_j^{(\ell-1)} \right) \right)$



Lecture-2: Overview

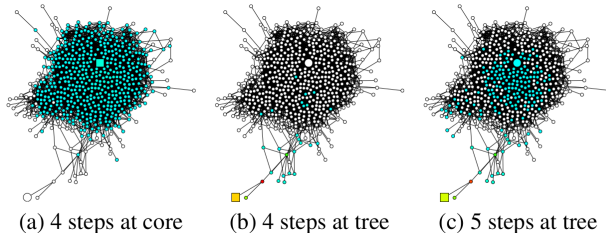
I Overview of GNN Landscape

- Learning on Graphs
- GraphSAGE
- **Jumping Knowledge Networks**
- Graph Isomorphism Network
- Graph Attention Network
- Relational GNNs



How to alleviate over-smoothing in GNNs?

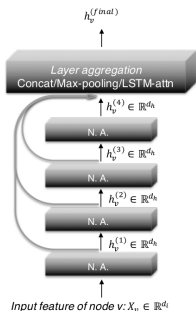
- GraphSAGE alleviates over-smoothing using *skip* connections (to some extent)
- Skip connection help only between successive layers
- Further, fixed neighbourhood sizes in sample and aggregate are restrictive



Expansion of a random walk (and hence influence distribution) starting at (square) nodes in subgraphs with different structures. Different subgraph structures result in very different neighborhood sizes.

Is it possible to learn the locality for each node and task?

- Architecture that selectively exploits information from neighbourhoods of differing locality
- Combines different aggregations at the last layer
- Representations *jump* to the last layer
- Jumping Knowledge Networks (JK-Nets)
- Similar to DenseNets!



Different Aggregators

- 1 Concatenation: $\text{CONCAT}[\mathbf{h}_i^{(1)}, \dots, \mathbf{h}_i^k]$
- 2 Max-pooling: $\text{MAX}[\mathbf{h}_i^{(1)}, \dots, \mathbf{h}_i^k]$
- 3 LSTM-attention based on attention scores derived from forward- and backward- LSTM features



Lecture-2: Overview

1 Overview of GNN Landscape

- Learning on Graphs
- GraphSAGE
- Jumping Knowledge Networks
- **Graph Isomorphism Network**
- Graph Attention Network
- Relational GNNs



GNNs and WL Algorithm

- GNNs operate by aggregating over neighbourhoods
- WL Algorithm relies on labelling based on neighbourhood
- Clear analogies between WL test and GNNs
- WL algorithm aggregates and updates discrete labels (with hash functions)
- GNNs aggregate and update node embeddings using neural networks
- GNNs as a continuous and differentiable analog of WL algorithm

Algorithm 1: GraphSAGE embedding generation (i.e., forward propagation) algorithm

Input : Graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$; input features $\{\mathbf{x}_v, \forall v \in \mathcal{V}\}$; depth K ; weight matrices $\mathbf{W}^k, \forall k \in \{1, \dots, K\}$; non-linearity σ ; differentiable aggregator functions $\text{AGGREGATE}_k, \forall k \in \{1, \dots, K\}$; neighborhood function $\mathcal{N}: v \rightarrow 2^{\mathcal{V}}$

Output : Vector representations \mathbf{z}_v for all $v \in \mathcal{V}$

```

1  $\mathbf{h}_v^0 \leftarrow \mathbf{x}_v, \forall v \in \mathcal{V}$ ;
2 for  $k = 1 \dots K$  do
3   for  $v \in \mathcal{V}$  do
4      $\mathbf{h}_{\mathcal{N}(v)}^k \leftarrow \text{AGGREGATE}_k(\{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\})$ ;
5      $\mathbf{h}_v^k \leftarrow \sigma(\mathbf{W}^k \cdot \text{CONCAT}(\mathbf{h}_v^{k-1}, \mathbf{h}_{\mathcal{N}(v)}^k))$ 
6   end
7    $\mathbf{h}_v^k \leftarrow \mathbf{h}_v^k / \|\mathbf{h}_v^k\|_2, \forall v \in \mathcal{V}$ 
8 end
9  $\mathbf{z}_v \leftarrow \mathbf{h}_v^K, \forall v \in \mathcal{V}$ 
```

Algorithm 1: WL-1 algorithm (Weisfeiler & Lehmann, 1968)

Input: Initial node coloring $(h_1^{(0)}, h_2^{(0)}, \dots, h_N^{(0)})$

Output: Final node coloring $(h_1^{(T)}, h_2^{(T)}, \dots, h_N^{(T)})$

$t \leftarrow 0$;

repeat

for $v_i \in \mathcal{V}$ **do**

$h_i^{(t+1)} \leftarrow \text{hash}(\sum_{j \in \mathcal{N}_i} h_j^{(t)})$;

$t \leftarrow t + 1$;

until stable node coloring is reached;

Formal connection between GNNs and WL Algorithm

There exists a GNN such that $\mathbf{h}_i^{(K)} = \mathbf{h}_j^{(K)}$ if and only if the two nodes i and j have the same label after K iterations of the WL algorithm



So, are all GNNs as powerful as the WL algorithm?

- Consider, a general message passing GNN:
$$\mathbf{h}_i^{(\ell)} = \text{UPDATE}^{(\ell)} \left(\mathbf{h}_i^{(\ell-1)}, \text{AGGREGATE} \left(\{\mathbf{h}_j^{(\ell-1)}, \forall j \in \mathcal{N}_i\} \right) \right)$$
- For GraphSAGE: $\mathbf{h}_i^{(\ell)} = \sigma \left(\Theta^{(\ell)} \text{CONCAT} \left(\mathbf{h}_i^{(\ell-1)}, \frac{1}{|\mathcal{N}_i|} \sum_{j \in \mathcal{N}_i} \mathbf{h}_j^{(\ell-1)} \right) \right)$

Conditions for GNNs to be as powerful as WL algorithm

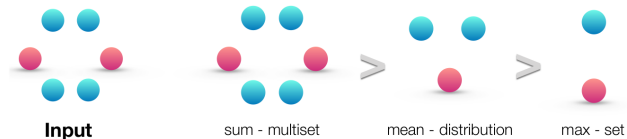
AGGREGATE and UPDATE functions must be injective i.e., they need to map every unique input to a unique output value.

What about GraphSAGE?

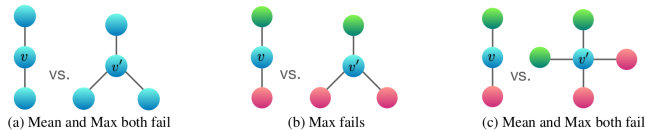
Mean operator is not injective!



Expressive power for different aggregators



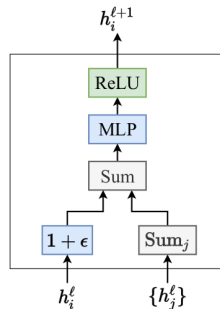
Sum captures the full multiset, mean captures the proportion/distribution of elements of a given type, and the max aggregator ignores multiplicities



Between the two graphs, nodes v and v' get the same embedding even though their corresponding graph structures differ.

Graph Isomorphism Network offers a fix!

- Propose a simple injective multiset function for AGGREGATE
- $\mathbf{h}_i^{(\ell)} = \text{MLP}^{(\ell)} \left(\left((1 + \epsilon^{(\ell)}) \cdot \mathbf{h}_i^{(\ell-1)} + \sum_{j \in \mathcal{N}_i} \mathbf{h}_j^{(\ell-1)} \right) \right)$, where $\epsilon^{(\ell)}$ is a trainable parameter.
- **Sum Aggregation:** Aggregates features from neighboring nodes.
- **Injective MLP:** Transforms aggregated features.
- **Learnable Epsilon (ϵ):** Provides flexibility in feature transformation.



But, is sum over multisets actually injective?

Sum Over Multisets

- Multiset: A generalization of a set that allows multiple instances of elements.
- Sum over multisets is injective if different multisets yield different sums.

$$\mathcal{X}_1 = \{2, 3, 3\}, \quad \sum_{x \in \mathcal{X}_1} x = 2 + 3 + 3 = 8$$

$$\mathcal{X}_2 = \{1, 2, 5\}, \quad \sum_{x \in \mathcal{X}_2} x = 1 + 2 + 5 = 8$$

- Example with numbers shows non-injectivity.
- High-dimensional feature vectors reduce collision chances.



Injectivity with Vectors

- Example with high-dimensional vectors:

$$\mathcal{X}_1 = \{(1, 0), (0, 1), (1, 1)\}$$

$$\sum_{x \in \mathcal{X}_1} x = (1, 0) + (0, 1) + (1, 1) = (2, 2)$$

$$\mathcal{X}_2 = \{(2, 1), (0, 0)\}$$

$$\sum_{x \in \mathcal{X}_2} x = (2, 1) + (0, 0) = (2, 1)$$

- Different vectors yield different sums.
- Sum aggregation effectively injective in high-dimensional space.



MLP as a Universal Approximator

- Multi-Layer Perceptron (MLP) can approximate any continuous function.
- Injective MLPs ensure unique transformations of aggregated features.

$$\mathbf{h}_i^{(\ell)} = \text{MLP}^{(\ell)} \left(\left(1 + \epsilon^{(\ell)} \right) \cdot \mathbf{h}_i^{(\ell-1)} + \sum_{j \in \mathcal{N}_i} \mathbf{h}_j^{(\ell-1)} \right)$$

- Ensures unique and expressive node representations.
- Enhances the discriminative power of GINs.



Lecture-2: Overview

I Overview of GNN Landscape

- Learning on Graphs
- GraphSAGE
- Jumping Knowledge Networks
- Graph Isomorphism Network
- **Graph Attention Network**
- Relational GNNs



Capturing Long-range Dependencies in GNNs

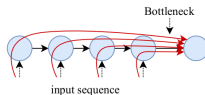
How many layers of a GNN are needed to receive information from other nodes at a radius of K

Let us look at an example

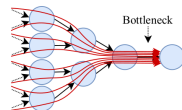


The Information Bottleneck Problem in GNNs

- With increasing depth, node embeddings converge to same values (over-smoothing)
- Many GNNs are unable to capture long-range dependencies
- Over-smoothing affects tasks with smaller *problem radii*
- Long-range dependencies correspond to larger *problem radii*
- As the number of layers increases, the number of nodes in each node's receptive field grows *exponentially*
- Results in *over-squashing*: Information from exponentially-growing receptive field is compressed into fixed length embeddings

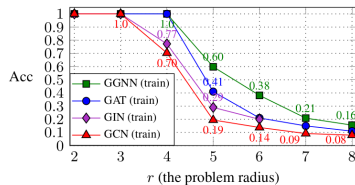
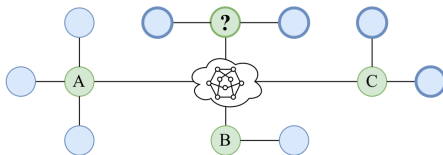


(a) The bottleneck of RNN seq2seq models



(b) The bottleneck of graph neural networks

Over-squashing in a controlled experiment



Solution to Over-Squashing?

Fully adjacent layer (fully connected graph in the last layer only)

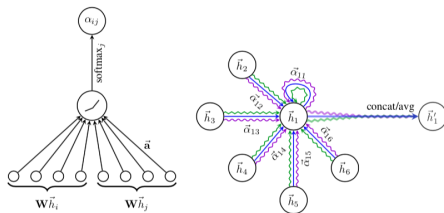
Neighbourhood Attention (unintended fix for over-squashing)

- All GNN methods (so far) give equal importance to all neighbours
- Recall, $\mathbf{h}_i^{(\ell+1)} = \text{UPDATE}^{(\ell)}\left(\mathbf{h}_i^{(\ell)}, \text{AGGREGATE}\left(\{\mathbf{h}_j^{(\ell)}, \forall j \in \mathcal{N}_i\}\right)\right)$
- Common AGGREGATE operators are sum, mean, max.
- Not all neighbours are equally important
- Attention as an importance weighting of neighbours:

$$\mathbf{h}_i^{(\ell)} = \sigma\left(\sum_{j \in \mathcal{N}_i} \alpha_{ij} \Theta^{(\ell)} \mathbf{h}_j^{(\ell-1)}\right)$$

$$\alpha_{ij} = \frac{\exp(\mathbf{a}^T [\Theta \mathbf{h}_i || \Theta \mathbf{h}_j])}{\sum_{k \in \mathcal{N}_i} \exp(\mathbf{a}^T [\Theta \mathbf{h}_i || \Theta \mathbf{h}_k])}$$

where $||$ is the concatenation operation.



Graph Attention using Multiple Heads

- Multiple attention *heads* can be used to learn differential aggregation
- Different heads can pay attention to different aspects of the graph structure

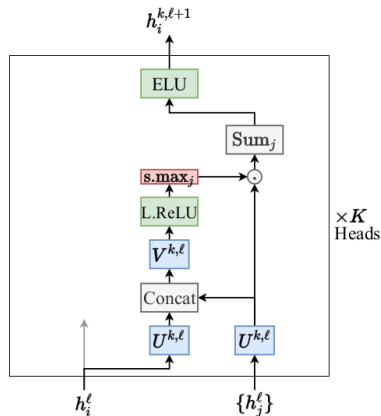


Figure 8: GAT Layer

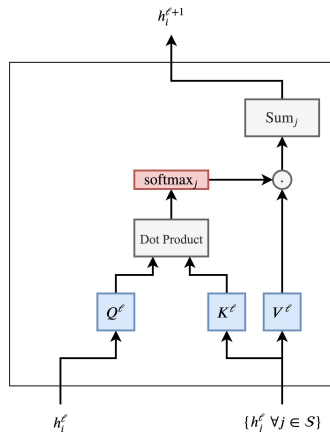
What about self-attention?

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

$$h_i^{\ell+1} = \text{Attention}\left(Q^\ell h_i^\ell, K^\ell h_j^\ell, V^\ell h_j^\ell\right)$$

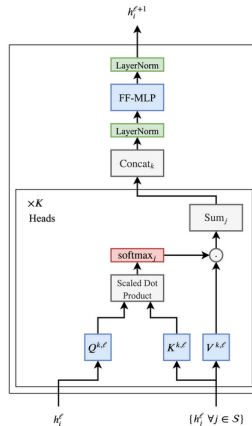
$$h_i^{\ell+1} = \sum_{j \in \mathcal{S}} \alpha_{ij} \left(V^\ell h_j^\ell\right)$$

$$\text{where, } w_{ij} = \text{softmax}\left(Q^\ell h_i^\ell \cdot K^\ell h_j^\ell\right)$$



Few more heads and tricks from GNNs to Transformers!!!

- Transformer and GAT with multiple heads are exactly equivalent if using a fully connected graph!
- Quadratic $\mathcal{O}(|\mathcal{V}|^2)$ time complexity of transformers arises from this
- GNNs have $\mathcal{O}(|\mathcal{V}||\mathcal{E}|)$ time complexity



Lecture-2: Overview

I Overview of GNN Landscape

- Learning on Graphs
- GraphSAGE
- Jumping Knowledge Networks
- Graph Isomorphism Network
- Graph Attention Network
- Relational GNNs



What about edge attributes?

- Neighbourhood aggregation is good for predictions on nodes or graph level
- Primary focus of GNNs is on aggregating from neighbourhood features
- Some of the methods can combine the information along the edges
- Mostly by weighting the adjacency matrix
- Explicit modelling of relations between nodes can be beneficial
- Tasks such as link prediction are important applications



Relational GNNs

- Explicit modeling of relations or edges:

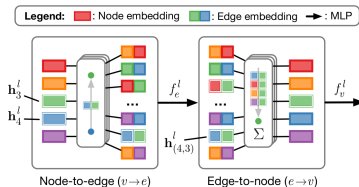
$$\text{Node embedding: } \mathbf{h}_j^{(\ell)} = g_n(\mathbf{h}_j^{(\ell-1)})$$

$$\text{Node-to-edge map: } \mathbf{h}_{(ij)}^{(\ell)} = g_{ne}([\mathbf{h}_i^{(\ell)}, \mathbf{h}_j^{(\ell)}])$$

$$\text{Edge-to-Node map: } \mathbf{h}_j^{(\ell+1)} = g_{en}\left(\sum_{i \in \mathcal{N}_j} \mathbf{h}_{ij}^{(\ell)}\right)$$

where $g(\cdot)$ are MLPs.

- Two types of messages bearing similarities with belief propagation



How to scale GNNs with growing scale of graphs!

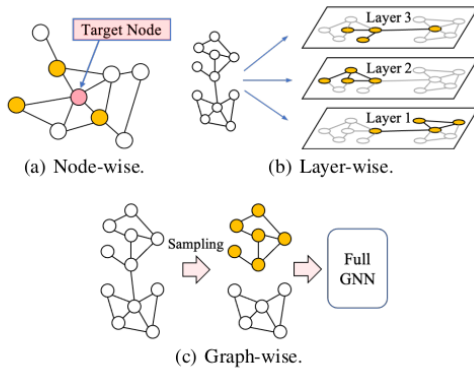
Category	Name	Scale	#Graphs	Average #Nodes	Average #Edges	Average Node Deg.
Node ogbn-	products	medium	1	2,449,029	61,859,140	50.5
	proteins	medium	1	132,534	39,561,252	597.0
	arxiv	small	1	169,343	1,166,243	13.7
	papers100M	large	1	111,059,956	1,615,685,872	29.1
	mag	medium	1	1,939,743	25,582,108	21.7

- Loosely, time and space complexity scales as $\mathcal{O}(N^2)$
- This can be a challenge for large/dense graphs
- Sparse representations alleviate to some extent; $\mathcal{O}(N)$
- More accurately,

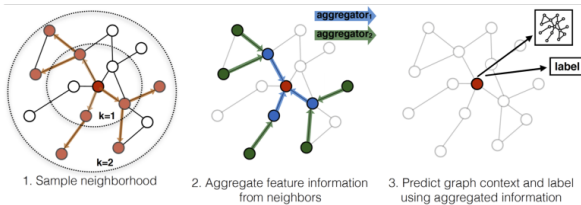
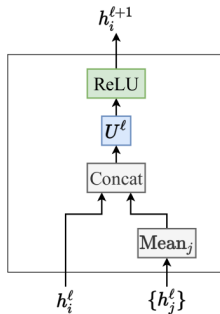
	Dense	Sparse
Forward Time	$LN^2F + LNF^2$	$LEF + LNF^2$
Forward Space	$N^2 + LF^2 + LNF$	$E + LF^2 + LNF$
Backward Time	$LN^2F + LNF^2$	$LEF + LNF^2$
Backward Space	$N^2 + LF^2 + LNF$	$E + LF^2 + LNF$



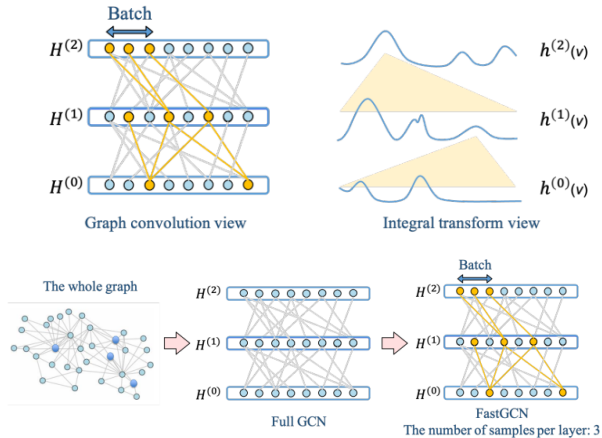
Resort to sub-sampling!



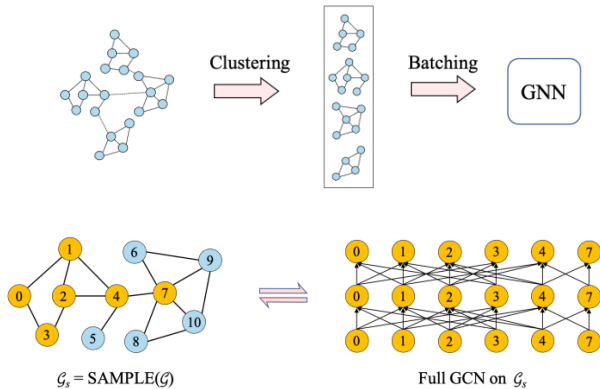
Node sampling based GNNs



Layer-wise sampling based GNNs



Graph-wise sampling based GNNs



Many other GNNs models arise based on sub-sampling!

Paradigm	Model	Sampling Strategy	Variance Reduction	Solved Problem	Characteristics
Node-wise Sampling	GraphSAGE (Hamilton et al 2017b)	Random	×	Inductive learning	Mini-batch training, reduce neighborhood expansion.
	VR-GCN (Chen et al 2018d)	Random	✓	Neighborhood expansion	Historical activations.
Layer-wise Sampling	FastGCN (Chen et al 2018c)	Importance	✓	Neighborhood expansion	Integral transform view.
	ASGCN (Huang et al 2018)	Importance	✓	Between-layer correlation	Explicit variance reduction, skip connection.
Graph-wise Sampling	Cluster-GCN (Chiang et al 2019)	Random	✓	Graph batching	Mini-batch on sub-graph.
	GraphSAINT (Zeng et al 2020a)	Edge Probability	✓	Neighborhood expansion	Variance and bias control.



Pooling methods for Graph ML

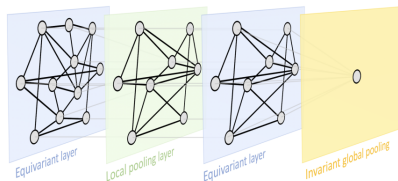
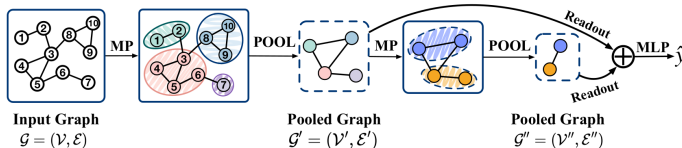


Figure 8: Geometric Deep Learning blueprint, exemplified on a graph. A typical Graph Neural Network architecture may contain permutation equivariant layers (computing node-wise features), local pooling (graph coarsening), and a permutation-invariant global pooling layer (readout layer).



Flat pooling methods based on Cluster Assignment Matrix (CAM)

Models	CAM Generator	Graph Coarsening
DiffPool ^[1]	$C = \text{softmax}(\text{GNN}_{\text{pool}}(\mathbf{X}, \mathbf{A}))$	$\begin{cases} \mathbf{X}' = \mathbf{C}^T \cdot \text{GNN}_{\text{emb}}(\mathbf{X}, \mathbf{A}) \\ \mathbf{A}' = \mathbf{C}^T \mathbf{A} \mathbf{C} \end{cases}$
NMF ^[2]	$\begin{cases} \mathbf{A} \approx \mathbf{U} \mathbf{V} \\ \mathbf{C} = \mathbf{V}^T \end{cases}$	$\mathbf{X}' = \mathbf{C}^T \mathbf{X}; \mathbf{A}' = \mathbf{C}^T \mathbf{A} \mathbf{C}$
LaPool ^[3]	$\begin{cases} s_i = \left\ \sum_{j \in \mathcal{N}(v_i)} A_{i,j} (\mathbf{x}_i - \mathbf{x}_j) \right\ _2 \\ \mathcal{V}_c = \{v_i \in \mathcal{V} \mid \forall v_j, s_i - \mathbf{A}_{ij} s_j > 0\} \\ \mathbf{C} = \text{sparsemax} \left(\mathbf{p} \frac{\mathbf{X} \mathbf{X}_c^T}{\ \mathbf{X}\ \ \mathbf{X}_c\ } \right) \end{cases}$	$\begin{cases} \mathbf{X}' = \text{MLP}(\mathbf{C}^T \mathbf{X}) \\ \mathbf{A}' = \mathbf{C}^T \mathbf{A} \mathbf{C} \end{cases}$
MinCut ^[4]	$\mathbf{C} = \text{MLP}(\mathbf{X})$	$\begin{cases} \mathbf{X}' = \mathbf{C}^T \mathbf{X}; \hat{\mathbf{A}} = \mathbf{C}^T \tilde{\mathbf{A}} \mathbf{C} \\ \mathbf{A}' = \hat{\mathbf{A}} - \mathbf{I} \text{diag}(\hat{\mathbf{A}}) \end{cases}$
StructPool ^[5]	$\mathbf{C} : \text{Minimize } E(\mathbf{C})^{\textcircled{5}}$	$\mathbf{X}' = \mathbf{C}^T \mathbf{X}; \mathbf{A}' = \mathbf{C}^T \mathbf{A} \mathbf{C}$
MemPool ^[6]	$\begin{cases} C_{i,j} = \frac{(1 + \ \mathbf{x}_i - \mathbf{k}_j\ ^2 / \tau)^{-\frac{\tau+1}{2}}}{\sum_{j'} (1 + \ \mathbf{x}_i - \mathbf{k}_{j'}\ ^2 / \tau)^{-\frac{\tau+1}{2}}} \\ \mathbf{C} = \text{softmax} \left(\Gamma \left(\left\ \begin{smallmatrix} m \\ C_t \end{smallmatrix} \right\ \right) \right) \end{cases}$	$\mathbf{X}' = \text{MLP}(\mathbf{C}^T \mathbf{X})$
HAP ^[7]	$\begin{cases} \mathbf{T} = \text{GCont}(\mathbf{X}) \\ C_{ij} = \sigma(\mathbf{p}^T [\mathbf{T}_{\text{Row}_i} \ \mathbf{T}_{\text{Col}_j}\]) \\ \mathbf{C} = \text{softmax}(\mathbf{C}) \end{cases}$	$\begin{cases} \mathbf{X}' = \text{MLP}(\mathbf{C}^T \mathbf{X}), \hat{\mathbf{A}} = \mathbf{C}^T \mathbf{A} \mathbf{C} \\ \mathbf{A}' = \text{Gumbel-SoftMax}(\hat{\mathbf{A}}) \end{cases}$
SEP ^[8]	$\mathbf{C} : \text{Minimize } \mathcal{H}^T(\mathcal{G})^{\textcircled{5}}$	$\mathbf{X}' = \mathbf{C}^T \mathbf{X}; \mathbf{A}' = \mathbf{C}^T \mathbf{A} \mathbf{C}$

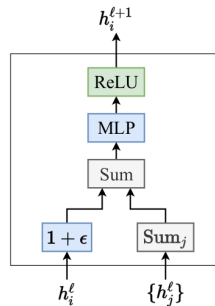
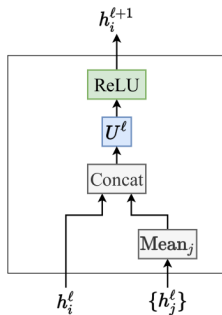
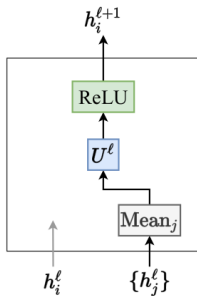


Hierarchical pooling methods based on Score Generators

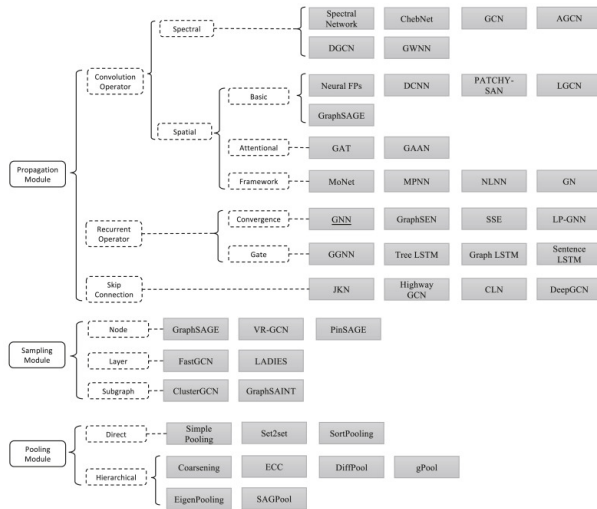
Models	Score Generator	Node Selector	Graph Coarsening
TopKPool ^[1]	$S = Xp / \ p\ _2$	$\text{idx} = \text{TOP}_k(S)$	$X' = X_{\text{idx}} \odot \sigma(S_{\text{idx}}); \quad A' = A_{\text{idx}, \text{idx}}$
SAGPool ^[2]	$S = \text{GNN}(X, A)$	$\text{idx} = \text{TOP}_k(S)$	$X' = X_{\text{idx}} \odot S_{\text{idx}}; \quad A' = A_{\text{idx}, \text{idx}}$
AttPool ^[3]	$S = \text{softmax}(XW)$	$\text{idx} = \text{TOP}_k(S)$	$X' = A_{\text{idx}}(X \odot S); \quad A' = A_{\text{idx}}AA_{\text{idx}}^T$
ASAP ^[4]	$S = \text{LEConv}(X^c, A)$	$\text{idx} = \text{TOP}_k(S)$	$X' = X_{\text{idx}}^c \odot S_{\text{idx}}; \quad A' = A_{\text{idx}}AA_{\text{idx}}^T$
HGP-SL ^[5]	$S = \ (I - D^{-1}A)X\ _1$	$\text{idx} = \text{TOP}_k(S)$	$\begin{cases} X' = X_{\text{idx}} \odot S_{\text{idx}}; \quad \hat{A} = A_{\text{idx}, \text{idx}} \\ A'_{ij} = \max(\sigma(\hat{a}[\mathbf{x}'(i, \cdot)]\ \mathbf{x}'(j, \cdot)]^T) + \lambda \cdot \hat{A}_{ij} \end{cases}$
VIPool ^[6]	$\begin{cases} P = \frac{1}{t} \sum_{h=1}^t (D^{-\frac{1}{2}} \tilde{A} D^{-\frac{1}{2}})^h W^h \text{MLP}(X) \\ S = \sigma(\text{MLP}(\text{MLP}(X, P))) \end{cases}$	$\text{idx} = \text{TOP}_k(S)$	$\begin{cases} X' = X_{\text{idx}} \odot S_{\text{idx}} \\ A' = \text{softmax}(A_{\text{idx}}) \text{softmax}(A_{\text{idx}})^T \end{cases}$
RepPool ^[7]	$S = \sigma(D^{-1}AXp / \ p\ _2)$	$\text{idx} = \text{SEL}_k(S)$	$\begin{cases} B = XW_{\text{idx}}(X_{\text{idx}})^T \\ X' = (\text{softmax}(B \odot M))^T (X \odot S) \\ A' = (\text{softmax}(B \odot M))^T A (\text{softmax}(B \odot M)) \end{cases}$
GSAPool ^[8]	$\begin{cases} S_1 = \text{GNN}(X, A) \\ S_2 = \sigma(\text{MLP}(X)) \\ S = \alpha S_1 + (1 - \alpha) S_2 \end{cases}$	$\text{idx} = \text{TOP}_k(S)$	$X' = (AXW)_{\text{idx}} \odot S_{\text{idx}}; \quad A' = A_{\text{idx}, \text{idx}}$
PANPool ^[9]	$\begin{cases} Z_i = \sum_{n=0}^L e^{-\frac{E(i)}{T}} \sum_{j=1}^N g(i, j; n) \\ M = Z^{-1} \sum_{n=0}^L e^{-\frac{E(n)}{T}} A^n \\ S = Xp + \beta \text{diag}(M) \end{cases}$	$\text{idx} = \text{TOP}_k(S)$	$X' = X_{\text{idx}} \odot \sigma(S_{\text{idx}}); \quad A' = A_{\text{idx}, \text{idx}}$
CGIPool ^[10]	$\begin{cases} S_r = \text{GNN}_r(X, A) \\ S_f = \text{GNN}_f(X, A) \\ S = \sigma(S_r + S_f) \end{cases}$	$\text{idx} = \text{TOP}_k(S)$	$X' = X_{\text{idx}} \odot S_{\text{idx}}; \quad A' = A_{\text{idx}, \text{idx}}$
TAPool ^[11]	$\begin{cases} S_l = \text{softmax}\left(\frac{1}{n}((XX^T) \odot (\tilde{D}^{-1}\tilde{A}))\mathbf{1}_n\right) \\ S_g = \text{softmax}(\tilde{D}^{-1}\tilde{A}Xp) \\ S = S_l + S_g \end{cases}$	$\text{idx} = \text{TOP}_k(S)$	$X' = X_{\text{idx}} \odot S_{\text{idx}}; \quad A' = A_{\text{idx}, \text{idx}}$



Summary: GCN, GraphSAGE, GIN



Other variations of GNN



Zhou et al. 2020 Graph neural networks: A review of methods and applications
 Vignac et al. 2023 DiGress: Discrete Denoising diffusion for graph generation

